CLAIMS

I/We claim:

[c1] 1. A method in a computing system for adapting saved states used with a first version of a game for use with a second version of the game, comprising:

comparing source code for the first version of the game to source code for the second version of the game to identify changes from the source code for the first version of the game to the source code for the second version of the game;

among the identified changes, selecting those that add a new dependency on state;

for each selected change, automatically generating a state modification rule that satisfies the new dependency added by the change;

storing the state modification rules together; and

in instructions used by the second version of the game to load saved states:

loading the current saved state;

determining whether the current state is used with the first version of the game; and

if the current state is used with the first version of the game:

retrieving the stored state modification rules;

in conjunction with loading the current state, applying the retrieved state modification rules to modify the loaded state; and

identifying the loaded state as being used with the second version of the game.

[c2] 2. A method in a computing system for adapting states used with a first version of a game for use with a second version of the game, comprising:

comparing the first and second versions of the game to identify dependencies on the state of the second version of the game not shared by the first version of the game;

automatically generating a rule to modify states used with the first version of the game to satisfy the identified dependency; and

for each of one or more states used with a first version of the game, applying the generated rule to the state.

[c3] 3. The method of claim 2 wherein the generated rule is applied to each state in response to an attempt to use the state with the second version of the game.

[c4] 4. A method in a computing system for analyzing a new version of a computer program, comprising:

comparing state data definitions of the new version of the computer program with state data definitions in a previous version of the computer program; and

if the comparison indicate that a data item moved in the state data hierarchy from an old location to a new location, generating an indication that, when state data generated by the previous version of the computer program is to be used with the new version of the computer program, the data item should be mapped from the old location to the new location.

[c5] 5. The method of claim 4, further comprising, if the comparison indicate that a data item in the state data definitions of the previous version of the computer program does not occur in the state data definitions of the new version of the computer program, generating an indication that, when state data

generated by the previous version of the computer program is to be used with the new version of the computer program, the data item should be represented in a different way in the state data for the new version of the computer program.

[c6] 6. The method of claim 4, further comprising, if the comparison indicate that a data item in the state data definitions of the new version of the computer program does not occur in the state data definitions of the previous version of the computer program, generating an indication that, when state data generated by the previous version of the computer program is to be used with the new version of the computer program, the data item should be assigned a starting value in the state data for the new version of the computer program..

[c7] 7. The method of claim 4 wherein the state data is stored on a heap.

[c8] 8. The method of claim 4 wherein the computer program is a game.

[c9] 9. The method of claim 4 wherein the data item is a global data item.

[c10] 10. The method of claim 4 wherein the data item is a data member of an object class.

[c11] 11. A computer-readable medium whose contents cause a computing system to adapt states used with a first version of a game for use with a second version of the game by:

comparing the first and second versions of the game to identify dependencies on the state of the second version of the game not shared by the first version of the game;

automatically generating a rule to modify states used with the first version of the game to satisfy the identified dependency; and

for each of one or more states used with a first version of the game,

applying the generated rule to the state.

[c12] 12. A computing system for analyzing a new version of a computer program, comprising:

a comparison subsystem that compares state data definitions of the new version of the computer program with state data definitions in a previous version of the computer program; and

a recommendation subsystem that, if the comparison indicate that a data item moved in the state data hierarchy from an old location to a new location, generates an indication that, when state data generated by the previous version of the computer program is to be used with the new version of the computer program, the data item should be mapped from the old location to the new location.

[c13] 13. A method in a computing system for generating rules relative to a game whose code base has been revised from an old version to a new version that has new dependencies on state, comprising:

constructing a rule that, when applied to a game state generated by the old version of the game's code base, will modify the game state to satisfy the new dependencies on state introduced by the new version of the game's code base;

storing the constructed rule; and

storing an indication that the stored rule is to be applied to a game state generated by the old version of the game's code base that are to be used with the new version of the game's code base.

[c14] 14. The method of claim 13, further comprising:

receiving user input; and

modifying the constructed rule in accordance with the user input before storing the constructed rule.

[c15] 15. The method of claim 13, further comprising, when the new version of the game's code base loads a state generated by the old version of the game's code base, applying the stored rule.

[c16] 16. The method of claim 15 wherein the stored rule is applied in the absence of user input.

[c17] 17. One or more computer memories collectively containing a game state update rule data structure, comprising:

(a) information usable to select one or more data items in a game state generated by a first version of game code; and

(b) information identifying a transformation on selected data items in a game state generated by the first version of game code, the transformation satisfying a dependency of a second version of game code on game state,

such that, when a distinguished game state generated by the first version of game code is to be used by the second version of game code, (a) can be used to select in the distinguished game state data items to be transformed, and (b) can be used to transform the selected data items to satisfy a dependency of a second version of game code on game state.

[c18] 18. The computer memories of claim 17 wherein (b) identifies a transformation that initializes a new component of the selected data items.

[c19] 19. The computer memories of claim 17 wherein (b) identifies a transformation that relocates the selected data items from an initial location within the game state to a new location within the game state.

[c20] 20. A method in a computing system for applying game state modification rules to game states used by a game, comprising:

when a game state is loaded by a present version of the game, reading an indication of the version of the game with which the game state is compatible;

if the read indication indicates that the game state is compatible with a version of the game earlier than the present version, modifying the loaded game state by:

selecting one or more game state modification rules indicated to transform game states compatible with the version of the game indicated by the read indication into game states compatible with the present version of the game; and

applying the selected game state modification rules to the loaded state.

[c21] 21. The method of claim 20, further comprising, if the read indication indicates that the game state is compatible with a version of the game earlier than the present version of the game, modifying the loaded game state by modifying the indication of the version of the game with which the game state is compatible to indicate that the game state is compatible with the present version of the game.

[c22] 22. The method of claim 20 or 21 wherein the loaded game state is only modified if the read indication indicates that the game state is compatible with a version of the game earlier than the present version of the game.

[c23] 23. The method of claim 20 wherein the selected game state modification rules include both (1) one or more game state modification rules indicated to transform game states compatible with the version of the game indicated by the read indication into game states compatible with an intermediate version of the game, and (2) one or more game state modification rules indicated to transform game states compatible with the intermediate version of the game into game states compatible with the present version of the game.

[c24] 24. A computer-readable medium whose contents cause a computing system to apply game state modification rules to game states used by a game, comprising:

when a game state is loaded by a present version of the game, reading an indication of the version of the game with which the game state is compatible;

if and only if the read indication indicates that the game state is compatible with a version of the game earlier than the present version:

selecting one or more game state modification rules indicated to transform game states compatible with the version of the game indicated by the read indication into game states compatible with the present version of the game; and

applying the selected game state modification rules to the loaded state; and

modifying the loaded game state by modifying the indication of the version of the game with which the game state is compatible to indicate that the game state is compatible with the present version of the game.

[c25] 25. A method of serializing a set of object instances residing on a heap, comprising:

visiting object instances residing on the heap by traversing the heap in breadth-first order;

for each object instance visited as part of the traversal, adding the class and data member values of the visited object instance to an intermediate data store; and

generating a stream representation of the set of object instances from the contents of the intermediate data store.

[c26] 26. The method of claim 25 wherein adding the class and data member values of the visited object instance to an intermediate data store comprises:

if the class identifier of the class of the visited object instance is not contained in a class table, reading a class index listed for the class identifier in the class table, and writing to an instance table the read class index and the data member values of the visited object instance;

if the class identifier of the class of the visited object instance is not contained in a class table, adding a class index to the class table for the class identifier, and writing to an instance table the added class index and the data member values of the visited object instance.

[c27] 27. The method of claim 26 wherein adding the class and data member values of the visited object instance to an intermediate data store comprises, if the class identifier of the class of the visited object instance is not contained in a class table, adding to a second class table:

the class index added to the class table for the class identifier,

a class identifier of the superclass of the class of the visited object instance; and

a table of the persistent fields of the class of the visited object instance.

[c28] 28. The method of claim 25, further comprising:

when the class and data member values of an object instance are added to the intermediate data store, setting an already-visited flag of the object instance,

and wherein the class and data member values of an visited object instance are added to the intermediate data store only if the already-visited flag of the object instance is not set.

[c29] 29. The method of claim 25 wherein the serialized object instances collectively represent the state of a game.

[c30] 30. One or more computer memories collectively containing a serialized object data structure representing a plurality of object instances, comprising:

a first table listing, for each class represented among the plurality of object instances:

an index for the class; and

a full name of the class;

a second table listing, for each class represented among the plurality of object instances:

a class index for the class;

a full name of a superclass of the class; and

a persistent field table identifying, for each persistent field of the class, a name of the persistent field and a type or class of the persistent field;

a third table listing, for each object instance among the plurality of object instances, the index of the class of the object instance; and

a linear dump of the values of the persistent fields for all object instances in the order of the third table.

[c31] 31. The computer memories of claim 30 wherein the linear dump is comprised of compact bytecode representations of the persistent field values.